Envoy Message Queuing

version 1.3

A Supplement to the Programmer's Guide for HP e3000

For use with Microsoft Message Queue services (MSMQ) software

ENVOY TECHNOLOGIES

Envoy MQ™, 2001

Envoy MQ Client for HP e3000, 2001

Internet: http://www.envoytech.com

Envoy MQ Client for HP e3000

Contents

Chapter 1

Installation

The Envoy MQ Client for HP e3000 is the component of Envoy MQ running on Hewlett-Packard HP e3000 platforms. The Envoy MQ Client communicates with the Envoy Message Queuing Connector (MQC), connecting your HP e3000 applications to the MSMQ network.

The Envoy MQ Client for HP e3000 is an extended version of Envoy MQ Client, which is described in the *Envoy MQ Programmer's Guide*. Envoy MQ Client for HP e3000 is specially adapted for programming in COBOL and C.

System and network requirements

You can install the Envoy MQ Client for HP e3000 on an HP e3000 system having the following minimum requirements:

- MPE/iX version 5.0 or higher.
- A TCP/IP communication link to at least one Windows NT system on which Envoy Message Queuing Connector (MQC) (version 1.3) is installed.

To install the software from the Envoy MQ CD-ROM, you need:

- A Windows system with a CD-ROM drive and an FTP connection to the HP e3000 system
- 5 Mb of free disk space for the Envoy MQ Client software

If you plan to use the COBOL programming interface described in Chapter 3, you need:

- HP COBOL II/XL compiler, version 3.15 or higher

Installation procedure

The following instructions are to install Envoy MQ Client on HP e3000 systems. You must also install Envoy Message Queuing Connector (MQC) on at least one Windows system in your network (for instructions, see the *Envoy Message Queuing Connector (MQC) Administrator's Guide*).

Where to install

You should install the Envoy MQ Client on each HP e3000 system that you want to connect to MSMQ.

Installation file

The Envoy MQ Client for HP e3000 software is distributed in a tar file called `HP3KTAR`, which is located in the `\clients\hp3000` directory on the Envoy MQ installation CD-ROM.

Procedure

Please follow the instructions below to install the Envoy MQ Client on HP e3000.

1. From a Windows system with a CD-ROM drive, transmit the `HP3KTAR` file to the HP e3000 by binary FTP or any equivalent method.
2. From an MPE prompt, run the following command to extract the contents of the tar file:

```
RUN TAR.HPBIN.SYS;INFO="-xvf HP3KTAR"
```

The contents of the tar file are extracted into a directory called `FMQCLIENT`. The following table contains a partial list of files and subdirectories in the `FMQCLIENT` directory.

File and directories	Description
`FMQDCCFG`	Envoy MQ configuration utility
`FMQVER`	Utility to display the Envoy MQ version information
`MSGTBL`	Message table containing error messages
`GWPING` `GWPONG`	Executable Envoy MQ test programs

File and directories	Description
FMQDCXL	Envoy MQ link library, in XL form for linking with COBOL programs
include/	Include files for C programs
lib/	Link library for C programs
samples/	C sample programs
COBOL/	COBOL sample programs and copybooks

Configuration

You need to configure the Envoy MQ Client parameters such as:

- Environment variables specifying the location of the Envoy MQ Client directory, log file, etc.
- The connection and logon information for Envoy Message Queuing Connector (MQC)
- A code page that Envoy MQ uses to translate string-valued message properties to UNICODE

For details of the configuration options and procedures, see the *Installation* chapter of the *Envoy MQ Client Programmer's Guide*. The following paragraphs contain a summary of the more important configurations.

Environment variables

You must set the following environment variable:

FMQROOT — The location of the Envoy MQ Client directory, where the main configuration file (FMQENV) is stored:

Optionally, you can set additional environment variables such as:

FMQOVERRIDE — The location of a supplementary configuration file.

FMQCONNECT	The name of a default Envoy Message Queuing Connector (MQC) connection.
FMQLOGPATH	The location of the debug log.
FMQDEBUG	Enables or disables debug logging.

Examples

```
SETVAR FMQROOT "/SYS/PUB/FMQCLIENT"
SETVAR FMQLOGPATH "CONSOLE"
SETVAR FMQDEBUG "ON"
```

For more information on the environment variables, see the *Envoy MQ Client Programmer's Guide*.

Configuration utility

From an MPE prompt, run the FMQDCCFG utility program to set the connection and code page parameters of Envoy MQ Client. For detailed instructions on using the utility, see the *Envoy MQ Client Programmer's Guide*.

Examples

The following examples illustrate the FMQDCCFG command-line syntax on the HP e3000.

- Define a connection to a Envoy Message Queuing Connector (MQC) located at the IP address 192.1.1.1, port 1100. In your programs, you can access the connection by the name newserver.

```
run /SYS/PUB/FMQCLIENT/FMQDCCFG;info="-SRV newserver -
NODE192.1.1.1 -PORT1100"
```

- Set newserver as the default connection:

```
run /SYS/PUB/FMQCLIENT/FMQDCCFG;info="-DEFnewserver"
```

- Set the Windows logon parameters for the newserver connection. (Omit this step is you connect to Envoy Message Queuing Connector (MQC) by the default logon method. See the *Programmer's Guide* for an explanation.)

```
run /SYS/PUB/FMQCLIENT/FMQDCCFG;info="-SRV newserver -
DMNEarth -USERJDoe -PWDTopSecret"
```

- Define a second Envoy Message Queuing Connector (MQC) connection called `server2`, whose parameters are identical to those of `newserver` except for the IP address.

```
run /SYS/PUB/FMQCLIENT/FMQDCCFG;info="-SRV server2 -
USESnewserver -node192.1.1.2"
```

- Download a translation table for code page 850 and store the table in a specified file:

```
run /SYS/PUB/FMQCLIENT/FMQDCCFG;info="-CP 850
/SYS/PUB/FMQCLIENT/CP850TBL"
```

- Set the default code page of `newserver` to 850.

```
run /SYS/PUB/FMQCLIENT/FMQDCCFG;info="-SRV newserver -
SCP850"
```

Installation test

To test the operation of Envoy MQ Client, run the `GWPING` and `GWPONG` programs supplied with the Envoy MQ software. These programs conduct a *ping-pong* test of the messaging system.

- The `GWPING` program sends *ping* messages via Envoy MQ Client and Envoy Message Queuing Connector (MQC) to a message queue.
- The `GWPONG` program sends *pong* replies to a second message queue, where it is read by `GWPING`.

✓ *Before you run the tests, you must define a default connection to Envoy Message Queuing Connector (MQC) and register the user name of the connection in Windows (for instructions, see* Configuration *on page 3**).*

Default test

In the default test, the `GWPING` program sends a sequence of ten test messages, each containing the text `"PING"`, to a queue called `.\PongQ`. The `GWPONG` program waits to receive the messages, and then sends them back to a queue called `.\PingQ`. The `GWPING` program reads the replies from `.\PingQ` and signals you when they are received.

Follow these steps to run the test.

1. Start the GWPONG program by entering the following command at an MPE prompt:

```
run /SYS/PUB/FMQCLIENT/GWPONG
```

2. Start the GWPING program with the following command:

```
run /SYS/PUB/FMQCLIENT/GWPING;info="-n 10"
```

For each of the ten test messages, GWPING should output *Ping sent* and *Received reply* together with the elapsed time.

In the event of an error, you should review the installation and configuration of the Envoy MQ Client and Envoy Message Queuing Connector (MQC).

Additional tests

You can set many test options for GWPING and GWPONG. For an explanation of the options, see the *Installation* chapter of the *Envoy MQ Programmer's Guide*.

Chapter 2

Programming Applications in C

The native language of the Envoy MQ API is C. The API is identical to the C-language API of other Envoy MQ Clients, and nearly identical to the API of MSMQ. Thus you can port MSMQ or Envoy MQ Client applications very easily from other platforms to HP e3000.

The following references provide further information on the API:

- For programming information, please see the *Programming Messaging Applications* chapter in the *Envoy MQ Programmer's Guide*.
- For details of the API syntax, you should have a copy of the Microsoft MSMQ documentation and SDK online help.

Header files

Include the Envoy MQ `wintypes.h` and `mq.h` headers in your program. The header files are located in the Envoy MQ `include` directory.

Compiling and linking

Link your program with the Envoy MQ `libfmqdc.a` library.

Source-code examples

For C source-code examples of Envoy MQ Client messaging applications, see the *Sample Application* chapter in the *Envoy MQ Programmer's Guide*. The source code of the `GWPING` and `GWPONG` programs is provided in the Envoy MQ `samples` directory.

Chapter 3

Programming Applications in COBOL

The Envoy MQ Client for HP e3000 provides a COBOL interface, which lets you call the Envoy MQ Client API functions directly from your COBOL programs. The interface provides all the needed COBOL definitions, so you can access the complete API without any C programming at all.

Operating system

The COBOL interface described in this chapter runs on MPE/iX version 5.0 or higher.

Overview of the interface

The interface is implemented as a set of external API procedures and copy members. This chapter explains:

- The steps for creating a Envoy MQ Client application in COBOL
- The structure and contents of `FMQCONST`, which is the most important of the copy members
- Techniques for calling the Envoy MQ Client API procedures

The interface provides two additional copy members, called `FMQPROPV` and `FMQLOC`, which support dynamic programming techniques for building message and queue property structures. The chapter includes:

- Sample COBOL data structures representing MSMQ message and queue properties, constructed using the dynamic techniques
- Sample COBOL messaging applications

API functions

This chapter describes an interface that you can use to call the Envoy MQ Client API functions in COBOL programs. It does not document the API functions themselves. For information on that subject, see the *Envoy MQ Programmer's Guide* and the Microsoft MSMQ documentation.

Programming steps

To program a Envoy MQ Client messaging application, follow these steps:

1. Copy the FMQCONST member, which is located in the Envoy MQ COBOL directory, into the working storage section of your COBOL program (see *FMQCONST copy member* below).
2. Optionally, copy the FMQPROPV and/or FMQLOC members (also in the COBOL directory) into the working storage section of your program. These members can help you set up the data structures you need for Envoy MQ Client API calls (see *Data structures* on page 21).
3. Create COBOL definitions for the required message and queue properties (see *Data structures* on page 21)..
4. Code the Envoy MQ Client API calls (see *Sample program* on page 30).
5. Compile the program using the HP COBOL II/XL compiler, version 3.15 or higher.
6. Using the MPE/iX Linkage Editor, link-edit the program with the Envoy MQ FMQDCXL library.

FMQCONST copy member

The FMQCONST copy member provides the definitions that you need to access the Envoy MQ Client API. You must copy FMQCONST into the working storage section of your COBOL program. FMQCONST is found in the Envoy MQ COBOL directory.

The FMQCONST definitions include:

- Constants representing message properties
- Constants representing queue properties
- Constants representing queue manager properties
- Constants representing the value types of properties
- Miscellaneous named constants

In general, the definitions are very similar to the C-language definitions in the C header files, `mq.h`, `wintypes.h`, and `fmqpubd.h`, which are also supplied with Envoy MQ Client. The main difference is that the COBOL identifiers contain hyphens (-) rather than underscores (_). For example, the C constant `PROPID_M_DEST_QUEUE_LEN` (representing the message property *destination queue name length*) is represented as `PROPID-M-DEST-QUEUE-LEN` in COBOL.

Message properties

The following table lists the message properties supported by MSMQ and Envoy MQ.

Information in the table

The table lists the following information about each property:

Property identifier — A constant (defined in `FMQCONST`) that identifies the property. When you use the property, you need to move this constant to the `aPropID` array of a property structure.

The identifiers are identical to the ones used in C, except that underscores (_) are replaced with hyphens (-).

Value type — A constant (defined in `FMQCONST`) that identifies the data type of the property value. When you use the property, you need to move this constant to the value type field of the propvariant array, in a property structure.

For a few properties, two value types are listed: the actual value type of the property, followed by the `VT-NULL` value type in parentheses. The `VT-NULL` type permitted only when receiving a message.

Data type of the value — The COBOL data type of the property value. When you use the property, you need to insert a value with the specified data type in the value field of the propvariant array, in a property structure.

For some properties, the value contains two fields. In those cases, the data types of both fields are listed.

For further explanation

For an explanation and examples of property structures, see *Property structure* on page 23. For an explanation of the aPropID and propvariant arrays, see *Substructures of property structures* on page 25).

For a list of the C data types corresponding to COBOL value types, see *Value type identifiers* on page 16).

A complete explanation of the meaning and use of each property is beyond the scope of this book. For that information, please see the Microsoft MSMQ documentation.

Property identifier	Value type identifier	Data type of the value
PROPID-M-ACKNOWLEDGE	VT-UI1 (or VT-NULL)	PIC X
PROPID-M-ADMIN-QUEUE	VT-LPWSTR	PIC S9(9) BINARY
PROPID-M-ADMIN-QUEUE-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-APPSPECIFIC	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-ARRIVEDTIME	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-AUTH-LEVEL	VT-UI4	PIC 9(9) BINARY
PROPID-M-AUTHENTICATED	VT-UI1 (or VT-NULL)	PIC X
PROPID-M-BODY	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY
PROPID-M-BODY-SIZE	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-BODY-TYPE	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-CLASS	VT-UI2 (or VT-NULL)	PIC 9(4) BINARY
PROPID-M-CONNECTOR-TYPE	VT-CLSID	PIC S9(9) BINARY
PROPID-M-CORRELATIONID	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY
PROPID-M-DELIVERY	VT-UI1 (or VT-NULL)	PIC X
PROPID-M-DEST-QUEUE	VT-LPWSTR	PIC S9(9) BINARY
PROPID-M-DEST-QUEUE-LEN	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-DEST-SYMM-KEY	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY

Property identifier	Value type identifier	Data type of the value
PROPID-M-DEST-SYMM-KEY-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-ENCRYPTION-ALG	VT-UI4	PIC 9(9) BINARY
PROPID-M-EXTENSION	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY
PROPID-M-EXTENSION-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-HASH-ALG	VT-UI4	PIC 9(9) BINARY
PROPID-M-JOURNAL	VT-UI1	PIC X
PROPID-M-LABEL	VT-LPWSTR	PIC S9(9) BINARY
PROPID-M-LABEL-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-MSGID	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY
PROPID-M-PRIORITY	VT-UI1 (or VT-NULL)	PIC X
PROPID-M-PRIV-LEVEL	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-PROV-NAME	VT-LPWSTR	PIC S9(9) BINARY
PROPID-M-PROV-NAME-LEN	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-PROV-TYPE	VT-UI4	PIC 9(9) BINARY
PROPID-M-RESP-QUEUE	VT-LPWSTR	PIC S9(9) BINARY
PROPID-M-RESP-QUEUE-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-SECURITY-CONTEXT	VT-UI4	PIC 9(9) BINARY
PROPID-M-SENDER-CERT	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY
PROPID-M-SENDER-CERT-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-SENDERID	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY

Property identifier	Value type identifier	Data type of the value
PROPID-M-SENDERID-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-SENDERID-TYPE	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-SENTTIME	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-SIGNATURE	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY
PROPID-M-SIGNATURE-LEN	VT-UI4	PIC 9(9) BINARY
PROPID-M-SRC-MACHINE-ID	VT-CLSID	PIC S9(9) BINARY
PROPID-M-TIME-TO-BE-RECEIVED	VT-UI4 (or VT-NULL)	PIC 9(9) BINARY
PROPID-M-TIME-TO-REACH-QUEUE	VT-UI4	PIC 9(9) BINARY
PROPID-M-TRACE	VT-UI1 (or VT-NULL)	PIC X
PROPID-M-VERSION	VT-UI4	PIC 9(9) BINARY
PROPID-M-XACT-STATUS-QUEUE	VT-LPWSTR	PIC S9(9) BINARY
PROPID-M-XACT-STATUS-QUEUE-LEN	VT-UI4	PIC 9(9) BINARY

Queue properties

The following table lists the queue properties supported by MSMQ and Envoy MQ.

For an explanation of the data listed in the table, see the table of *Message properties* above.

Property identifier	Value type identifier	Data type of the value
PROPID-Q-AUTHENTICATE	VT-UI1	PIC X
PROPID-Q-BASEPRIORITY	VT-I2	PIC S9(4) BINARY

Property identifier	Value type identifier	Data type of the value
PROPID-Q-CREATE-TIME	VT-I4	PIC S9(9) BINARY
PROPID-Q-INSTANCE	VT-CLSID	PIC S9(9) BINARY
PROPID-Q-JOURNAL	VT-UI1	PIC X
PROPID-Q-JOURNAL-QUOTA	VT-UI4	PIC 9(9) BINARY
PROPID-Q-LABEL	VT-LPWSTR	PIC S9(9) BINARY
PROPID-Q-MODIFY-TIME	VT-I4	PIC S9(9) BINARY
PROPID-Q-PATHNAME	VT-LPWSTR	PIC S9(9) BINARY
PROPID-Q-PRIV-LEVEL	VT-UI4	PIC 9(9) BINARY
PROPID-Q-QUOTA	VT-UI4	PIC 9(9) BINARY
PROPID-Q-TRANSACTION	VT-UI1	PIC X
PROPID-Q-TYPE	VT-CLSID	PIC S9(9) BINARY

Queue manager properties

The following table lists queue manager properties supported by MSMQ and Envoy MQ.

For an explanation of the data listed in the table, see the table of *Message properties* above.

Property identifier	Value type identifier	Data type of the value
PROPID-QM-CONNECTION	VT-VECTOR-LPWSTR	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY
PROPID-QM-ENCRYPTION-PK	VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY
PROPID-QM-MACHINE-ID	VT-CLSID	PIC S9(9) BINARY
PROPID-QM-PATHNAME	VT-LPWSTR	PIC S9(9) BINARY

Property identifier	Value type identifier	Data type of the value
PROPID-QM-SITE-ID	VT-CLSID	PIC S9(9) BINARY

Value type identifiers

The following table lists the value type identifiers defined in FMQCONST and the corresponding identifiers defined in the C header files. Only the identifiers that are currently used in MSMQ are listed.

The value types are used in propvariant structures, which store the values of properties. For a full explanation, see *Substructures of property structures* on page 25. For reference, the table also indicates:

- The data types of the value fields in a propvariant structure
- The suggested data names for the property values
- The interpretation of the value fields
- The names of the corresponding value fields in C

COBOL				Equivalent in C		
Value type identifier	Data type of the value	Suggested data names[c]	Interpretation of property value	Value type identifier	Data type of the value	Union field name
VT-CLSID	PIC S9(9) BINARY	MQ-PUUID	Base pointer (points to a GUID code)	VT_CLSID	CLSID _RPC_FAR	*puuid
VT-I2	PIC S9(4) BINARY	MQ-IVAL	Property value	VT_I2	short	iVal
VT-I4	PIC S9(9) BINARY	MQ-LVAL	Property value	VT_I4	long	lVal
VT-LPWSTR	PIC S9(9) BINARY	MQ-LPWSTR	Base pointer (points to a null-terminated string)	VT_LPWSTR	LPWSTR	pwszVal

COBOL				Equivalent in C		
Value type identifier	**Data type of the value**	**Suggested data names[c]**	**Interpretation of property value**	**Value type identifier**	**Data type of the value**	**Union field name**
VT-NULL			No value (permitted only when receiving a message)	VT_NULL		
VT-UI1	PIC X	MQ-BVAL	Property value	VT_UI1	UCHAR	bVal
VT-UI2	PIC 9(4) BINARY	MQ-UIVAL	Property value	VT_UI2	USHORT	uiVal
VT-UI4	PIC 9(9) BINARY	MQ-ULVAL	Property value	VT_UI4	ULONG	ulVal
VT-VECTOR-LPWSTR	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY	MQ-CALPWSTR-CELEMS MQ-CALPWSTR-PELEMS	Length of buffer Base pointer (points to buffer)[a]	VT_VECTOR | VT_LPWSTR	CALPWSTR	calpwstr
VT-VECTOR-UI1	Two fields: PIC S9(9) BINARY PIC S9(9) BINARY	MQ-CAUB-CELEMS MQ-CAUB-PELEMS	Length of buffer Base pointer (points to buffer)[b]	VT_VECTOR | VT_UI1	CAUI1	caub

Notes

a. For the value type VT-VECTOR-LPWSTR, the buffer contains a null-terminated string.

b. For the value type VT-VECTOR-UI1, the buffer may contain various types of binary or text data:

- The message body property (PROPID-M-BODY) has this value type and may contain any data whatsoever.
- Other properties having this value type are restricted to certain types or structures of data. For information about specific properties, see the Microsoft MSMQ documentation and SDK online help.

c. The data names are defined in the FMQPROPV copy member. You can replace the MQ- prefix with another prefix when you copy FMQPROPV into your program.

Miscellaneous named constants

FMQCONST defines a large number of constants representing special values of API function arguments, error codes, etc. The following are a few examples:

Constant in COBOL	Equivalent in C
MQ-ACCESS-ALL	PSD_SPECIALACCESS_ALL
MQ-ERROR-ACCESS-DENIED	MQ_ERROR_ACCESS_DENIED
MQ-ERROR-BUFFER-OVERFLOW	MQ_ERROR_BUFFER_OVERFLOW
MQ-LE	PRLE

✓ *The constants are too numerous to list here. For a complete listing, please refer to the FMQCONST source code.*

API functions

The COBOL interface provides a complete set of definitions for the Envoy MQ Client API functions. The functions are called as external procedures in COBOL.

Calling syntax

In the procedure section of your program, you can call the `MQSendMessage` procedure using syntax such as the following. The procedure is equivalent to the `MQSendMessage()` function in the MSMQ or Envoy MQ Client API.

```
CALL 'MQSendMessage' USING
                                \Queue-Handle\
                BY REFERENCE Props
                                \pTransaction\
                RETURNING MQ-Result-Long.
EVALUATE MQ-Result
    WHEN MQ-OK    GO TO Send-Message-Exit
    WHEN OTHER    DISPLAY ERR-MSG
                  PERFORM Envoy MQ-Disconnect
END-EVALUATE.
```

The procedure accepts three parameters:

`Queue-Handle`	Specifies the destination queue.
`Props`	A message property structure, containing the content of the message.
`pTransaction`	A transaction handle of type A(16), specifying a transaction to which the message belongs (optionally `NULL`).

The procedure returns a numerical result code `MQ-Result-Long`.

Comparison with C

For comparison, the following is the corresponding API function declaration in C:

```
HRESULT APIENTRY MQSendMessage(
  QUEUEHANDLE hDestinationQueue,
  MQMSGPROPS * pMessageProps,
  ITransaction * pTransaction
);
```

Samples of other API calls

For other examples of COBOL API calls, see the *Sample program* on page 30.

For more examples, see the *Online samples* listed on page 41. In the online samples, you can find examples of many Envoy MQ Client API procedures including:

- Setting up the input parameters of each procedure
- The syntax for the procedure call
- Interpreting the output parameters and return values

List of Envoy MQ Client procedures

The following is a list of Envoy MQ Client API procedures. The table includes:

- The COBOL procedure names, which are identical to the C function names
- The Envoy MQ Client sample programs where the API calls are illustrated (see *Online samples* on page 41)
- References for additional information, including a complete explanation of each procedure and its parameters.

The key for the additional references is as follows:

A. The chapter on *Programming Messaging Applications* in the *Envoy MQ Programmer's Guide*.

B. The Microsoft MSMQ documentation and SDK online help

Procedure	Sample programs where illustrated	Additional references
FMQAbort	FMQBSAMP	A
FMQCommit	FMQBDYN	A
FMQConnect	FMQBSAMP	A
FMQDebug	FMQBSAMP	A
FMQDisconnect	FMQBSAMP	A
FMQGetLogPath	FMQBSAMP	A
FMQSetLogPath	FMQBSAMP	A
FMQVersion	FMQBSAMP	A
FMQV1Connect	FMQBSAMP	A
MQBeginTransaction	FMQBDYN	A
MQCloseCursor	FMQBSAMP	B
MQCloseQueue	FMQBDYN, FMQBSTC	B
MQCreateCursor	FMQBSAMP	B
MQCreateQueue	FMQBDYN, FMQBSTC	A, B
MQDeleteQueue	FMQBDYN, FMQBSTC	B
MQFreeMemory	FMQBLOC	B

Procedure	Sample programs where illustrated	Additional references
MQFreeSecurityContext	FMQBDYN	A, B
MQGetMachineProperties	FMQBSAMP	B
MQGetQueueProperties	FMQBSAMP	B
MQGetSecurityContext	FMQBDYN	A, B
MQHandleToFormatName	FMQBSAMP	B
MQInstanceToFormatName	FMQBSAMP	B
MQLocateBegin	FMQBLOC	A, B
MQLocateEnd	FMQBLOC	B
MQLocateNext	FMQBLOC	B
MQOpenQueue	FMQBDYN, FMQBSTC	B
MQPathNameToFormatName	FMQBDYN, FMQBSTC	B
MQReceiveMessage	FMQBDYN, FMQBSTC	A, B
MQRegisterCertificate	FMQBSAMP	A, B
MQSendMessage	FMQBDYN, FMQBSTC	B
MQSetQueueProperties	FMQBSAMP	B

Data structures

Many of the MSMQ and Envoy MQ Client API functions require parameters that are pointers to data structures. These include:

Property structures — Structures containing sets of message, queue, or queue manager properties. The content of a message, for example, is specified in a message property structure.

Substructures of property structures — Structures and arrays that are elements of property structures. An example is the *propvariant structure*, which contains the values of properties.

Query structures Structures required as parameters of the `MQLocateBegin` function, which searches for queues having specified property values.

This section explains how you can create the property structures and substructures in your COBOL programs. If you wish, you can copy the examples (with minor modifications) into your COBOL programs. You can find additional examples in the *Sample program* on page 30.

For additional information on the interpretation and use of the structures, please refer to the Microsoft MSMQ documentation and SDK online help.

For information on the query structures, please see the *Online samples* described on page 41.

Programming method

Suppose that your application creates a queue and sends and receives messages containing various sets of message properties. Before you call the `MQCreateQueue` API function, you need to create a queue property structure including several queue properties. Before you call `MQSendMessage` and `MQReceiveMessage`, you need to create a message property structure containing the message properties.

In a COBOL program, you can implement the property structure using arrays or multiple-occurrence data structures. In the definition specifications, you need to define the maximum size of the arrays or the maximum number of occurrences. You also need to define pointers to the first element or occurrence.

In the procedure division, the program sets the number of active array elements or occurrences, that is, the number of properties included in the structure. The program then moves the desired queue or message properties into the arrays or structures.

In this way, the program can change the set of properties dynamically, before each Envoy MQ Client API call.

✓ *As an alternative to the above dynamic method, you can create static data structures in the definitions specifications. For an example using static structures, see the `FMQBSTC` sample program in the `COBOL` directory (see Online samples on page 41).*

Property structure

A *property structure* contains a collection of properties and their values. There are three types of property structures, which have different C data types.

Structure	Contains a collection of	C data type
Message property structure	Message properties	MQMSGPROPS
Queue property structure	Queue properties	MQQUEUEPROPS
Queue manager property structure	Queue manager properties	MQQMPROPS

Each property structure contains the following four fields:

COBOL data type	C data type	Field name in C	Description
PIC 9(9) BINARY	DWORD	cProp	A count of the properties included in the structure. The value of this field is the size of the arrays in the other fields of the structure.
PIC S9(9) BINARY	Array of PROPID	aPropID	A pointer to an array of PROPID-... constants, identifying the properties that are included in the structure (input to the API functions).
PIC S9(9) BINARY	Array of PROPVARIANT	aPropVar	A pointer to an array of propvariant structures, which contain the values of the properties (input or output).
PIC S9(9) BINARY	Array of HRESULT	aStatus	A pointer to an array of status codes (output from the API functions).

✓ *In the following discussion, we refer to the fields by their generic names cProp, aPropID, etc. In COBOL, you must use field names that are unique throughout the entire program.*

The three types of property structures all contain the same four fields. This means that you can represent them in COBOL by defining a single top-level property structure. To create a message property structure, you can store pointers to arrays of message properties in the fields. To create a queue or

queue-manager property structure, you can store pointers to arrays of queue properties or queue-manager properties in the fields.

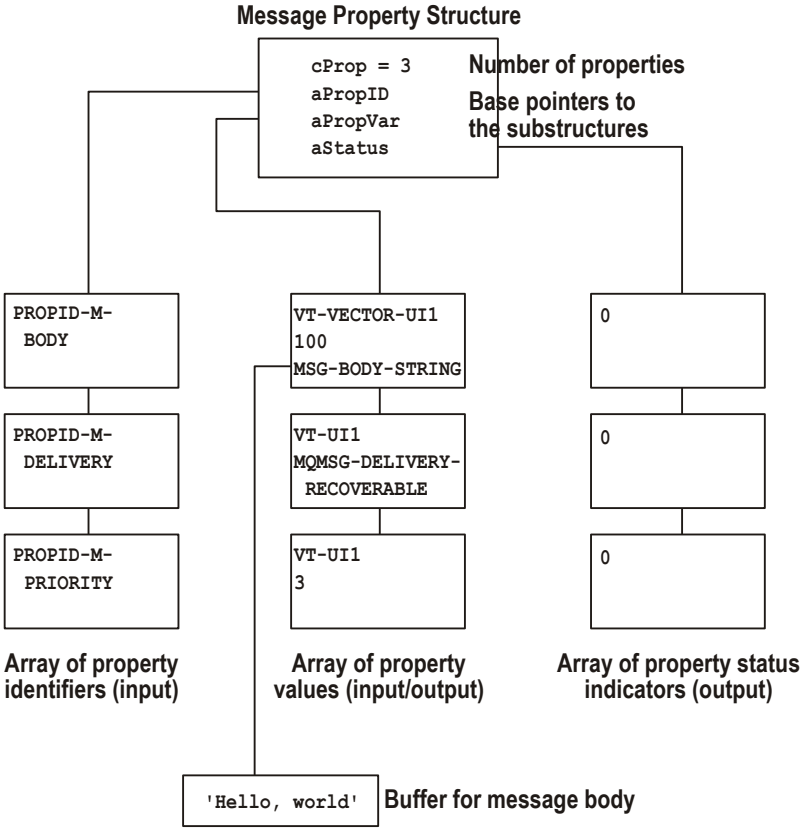
The following is a sample definition of the property structure:

```
* Top level property structure
 01 Props.
    02 cProp           PIC 9(9) BINARY.
    02 aPropID         PIC S9(9) BINARY.
    02 aPropVar        PIC S9(9) BINARY.
    02 aStatus         PIC S9(9) BINARY.
```

Substructures of property structures

The property structure contains pointers to three arrays:

`aPropID`	Pointer to an array of property identifies (`PROPID-...` constants) identifying message, queue, or queue manger properties.
`aPropVar`	Pointer to an array of propvariant structures, which contain the values of the properties.
`aStatus`	Pointer to an array of status codes, used for output from the API functions.



The number of elements in each array is given by the `cProp` field of the property structure. The order of properties must be identical in each array.

For example, if the aPropID array contains PROPID-... constants for the message body, delivery, and priority properties, then the other arrays must also contain elements for exactly the same properties, in the same order.

The following example illustrates how you can construct the arrays in a COBOL program. For convenience, the arrays are represented as multiple-occurrence data structures (in essence, substructures of a property structure) instead of true COBOL arrays.

The example is for a message property structure containing a maximum of 10 properties. We will use the property structure to construct a message containing three properties:

- Message body
- Message delivery
- Message priority

The other seven properties in the property structure are not used in this example.

Setting the number of active properties

The number of properties in the property structure is stored in the cProp field of the property structure. In the sample message, there are three properties. You can specify this in the procedure division by writing:

```
MOVE 3 TO cProp.
```

This instructs Envoy MQ Client to use the first three properties of the property structure. If any additional properties exist in the structure, they are ignored.

If you later need a property structure containing a different number of properties, you can reset cProp to the new value, up to the array size of the property structure.

Array of property identifiers

The array of property identifiers corresponds to the aProp field of a property structure in C. In COBOL, you can define the array as follows:

```
* aPropID array of up to 10 property identifiers
 01 MQ-PropID-Array.
    02 MQ-PropID PIC 9(9) BINARY OCCURS 10.
```

Here, we have defined the array size for a maximum of 10 properties. Only three of the properties are used in the message example.

In the procedure division, we need to:

- Set the `aPropID` pointer of the property structure to point to the array
- Move the property identifiers to the array

For our sample message, we would write:

```
* Set the aPropID pointer of the property structure
 CALL INTRINSIC ".LOC." USING MQ-PropID-Array GIVING
aPropID.
*
* Move the property identifiers to the array
 MOVE PROPID-M-BODY              TO MQ-PropID(1).
 MOVE PROPID-M-DELIVERY          TO MQ-PropID(2).
 MOVE PROPID-M-PRIORITY          TO MQ-PropID(3).
```

Array of propvariant structures

MSMQ and Envoy MQ Client use propvariant structures to store the values of message, queue, and queue manager properties. In HP e3000, a propvariant is a 16-byte structure containing the following fields:

Value type identifier	A `VT-...` constant indicating the data type of the property value.
Reserved	Reserved for future use.
Value1	The value of the property. For certain properties, *Value1* is the size of the value in bytes (equivalent to the `cElems` field in C).
Value2	If *Value1* contains the value, *Value2* is an empty placeholder field. If *Value1* contains the size of the value, then *Value2* is a pointer to the value (equivalent to the `pElems` field in C).

In COBOL, you can define the array of propvariant structures as a multiple-occurrence data structure. The elements of the structure are copies of the `FMQPROPV` member, which is supplied in the Envoy MQ `COBOL` directory. `FMQPROPV` contains a complete COBOL definition of the propvariant data structure.

```
* aPropVar array of up to 10 property values
 01 MQ-PropVar-Array.
    02 MQ-PropVar OCCURS 10.
       COPY FMQPROPV REPLACING ==:MQ:== BY ==MQ==.
```

✓ *You can define more than one `aPropVar` array using the `FMQPROPV` copy member. In each array, copy `FMQPROPV` replacing `:MQ:` with a different string, such as `MQ1`, `MQ2`, etc.*

In the procedure division, we need to:

- Set the `aPropVar` pointer of the property structure to point to the array
- Move the appropriate value type identifier, *Value1*, and *Value2* for each message property, to the first three elements of the array

The *Value1* and *Value2* fields in `FMQPROPV` have different names and data types depending on the property that you want to store. The names are illustrated in the sample code below. For a complete listing of the *Value* names, see the table of *Value type identifiers* on page 16.

```
* Set the aPropVar pointer of the property structure
 SET aPropVar TO ADDRESS OF MQ-PropVar-Array.
*
* Set the message body to a 'Hello, World' test string
     MOVE VT-VECTOR-UI1   TO MQ-VARTYPE(1).
* Value1 of the message body property is the length of
the body
     MOVE 12              TO MQ-CAUB-CELEMS(1).
* Value2 is a pointer to a buffer containing the message
body
     SET MQ-CAUB-PELEMS(1) TO ADDRESS OF MSG-BODY-
STRING.
*
* Set the delivery property to recoverable
     MOVE VT-UI1          TO MQ-VARTYPE(2).
* Value1 of the delivery property (there is no Value2)
     MOVE MQMSG-DELIVERY-RECOVERABLE TO MQ-BVAL(2).
*
* Set the priority property to a value of 3
     MOVE VT-UI1          TO MQ-VARTYPE(3).
* Value1 of the priority property (there is no Value2)
     MOVE 3               TO MQ-BVAL(3).
```

Elsewhere in the program, you need to define a buffer and store the message in body in it, for example:

```
* Buffer containing a test message body
 77 MSG-BODY-STRING    PIC X(50) VALUE 'Hello, world'.
```

Array of status codes

The array of status codes corresponds to the `aStatus` field in C. A sample definition follows:

```
       01 MQ-Prop-Result-Array.
          02 MQ-Prop-Result PIC 9(9) BINARY OCCURS 10.
```

The status codes are output from various API functions. In the procedure division, you need to set the `aStatus` pointer in the property structure to the address of the array:

```
 CALL INTRINSIC ".LOC." USING MQ-Prop-Result-Array
GIVING aStatus.
```

String handling

Several of the message, queue, and queue manager properties have values that are character strings. For example, the message label is a string of up to 250 characters. In addition, certain Envoy MQ Client API functions (for example `FMQConnect`), require parameters that are strings.

This section describes the differences between C and COBOL strings and the steps to ensure compatibility of your programs with the MSMQ standard.

✓ *For details of the maximum string length, etc., see the Microsoft MSMQ documentation and SDK online help.*

Null-terminated strings

MSMQ and Envoy MQ Client require that every string value be terminated by a null character. In COBOL, strings are predefined in length and are padded with trailing blanks. You can convert strings between the two formats using the COBOL built-in function `STRING`.

UNICODE conversion

Envoy MQ Client uses a code-page translation table to translate string properties and parameters from UNICODE or vice versa.

All message and queue properties are converted, with the following exceptions:

- The message body (`PROPID-M-BODY`) is converted only if the message body type (`PROPID-M-BODY-TYPE`) is `VT-LPWSTR` or `VT-BSTR`. Envoy MQ does not translate a message body of any other type because it doesn't know whether the body contains text or binary data. Instead, you should program whatever conversions are needed.
- The message extension (`PROPID-M-EXTENSION`).

Sample program

This section presents the complete source code of the `FMQBDYN` sample program, which is supplied online in the Envoy MQ `COBOL` directory. The program illustrates some basic messaging operations, including:

- Creating and deleting a queue
- Converting a queue path name to a format name
- Opening and closing a queue
- Sending and receiving messages
- Working with the MSMQ message authentication service
- Working with transactions

The program uses the dynamic method to create the required MSMQ and Envoy MQ Client data structures. For a detailed discussion of the structures, see *Data structures* on page 21.

✓ *For additional sample programs, see* Online samples *on page 41.*

Source code

```
 IDENTIFICATION DIVISION.
    PROGRAM-ID. 'FMQBDYN'.
*****************************************************************
*                                                               *
* Description: Sample COBOL program demonstrating the           *
*              use of dynamic property structures and the       *
*              FMQCONST and FMQPROPV copy members               *
*                                                               *
* Ver: 1.2                                                      *
*                                                               *
* Envoy MQ Client for HP e3000                                   *
* (C) Copyright 2001 by Envoy Technologies, Inc.                    *
* All rights reserved                                           *
*                                                               *
*****************************************************************
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
* Include Envoy MQ definitions in the program
*
     COPY FMQCONST.
*
* aPropID array of up to 10 property identifiers
 01 MQ-PropID-Array.
    02 MQ-PropID PIC 9(9) BINARY OCCURS 10.
*
* aPropVar array of up to 10 property values
* Note : This sample uses the same property structure for both
*        queue and message properties. You may define additional
*        property structures using the COPY REPLACING feature.
*
 01 MQ-PropVar-Array.
    02 MQ-PropVar OCCURS 10.
```

```
       COPY FMQPROPV REPLACING ==:MQ:== BY ==MQ==.
*
* aStatus array of up to 10 property status codes
 01 MQ-Prop-Result-Array.
    02 MQ-Prop-Result PIC 9(9) BINARY OCCURS 10.
*
* Top level property structure
 01 Props.
    02 cProp           PIC 9(9) BINARY.
    02 aPropID         PIC S9(9) BINARY.
    02 aPropVar        PIC S9(9) BINARY.
    02 aStatus         PIC S9(9) BINARY.
*
 01 MQ-Result          PIC S9(9) BINARY.
 01 FILLER REDEFINES MQ-Result.
    05 MQ-Result-Flags PIC S9(4) BINARY.
    05 MQ-Result-Seq   PIC 9(4) BINARY.
*
 77 FormatName-Length  PIC 9(9) BINARY.
 77 Queue-Handle       PIC 9(9) BINARY.
 77 Connection-Handle  PIC 9(9) BINARY VALUE 0.
 77 SecContext-Handle  PIC 9(9) BINARY.
 77 pTransaction       PIC 9(9) BINARY.
 77 FormatName         PIC X(125).
 77 Q-PATH-STRING      PIC X(125).
 77 Q-LABEL-STRING     PIC X(125).
 77 MSG-COUNTER        PIC 9(3).
 77 MSG-BODY-STRING    PIC X(50).
 77 MSG-BODY-PREFIX    PIC X(18) VALUE  'Message Number is '.
 77 MSG-LABEL-STRING   PIC X(22).
 77 MSG-LABEL-PREFIX   PIC X(18) VALUE 'Message Label is '.
 77 ERR-MSG      PIC X(23) VALUE 'Envoy MQ call failed!'.
 77 AUTH-Msg     PIC X(30) VALUE 'Authenticated message received'.
 77 NOT-AUTH-Msg PIC X(30) VALUE 'Unauthenticated message! '.
 77 WS-END       PIC X.
*-----------------------------------------------------------------
 PROCEDURE DIVISION.
*
 Main SECTION.
*
 Main-P.
*
* Set the pointers of the property structure. The same structure is
* used for both queue and message properties.
     CALL INTRINSIC ".LOC." USING MQ-PropID-Array
          GIVING aPropID.
     CALL INTRINSIC ".LOC." USING MQ-PropVar-Array
```

```
          GIVING aPropVar.
     CALL INTRINSIC ".LOC." USING MQ-Prop-Result-Array
          GIVING aStatus.
*
* Initialize error flags for error checking
     COMPUTE MQ-ERROR = -16370.
*
* Begin
*
* Create a queue if it doesn't already exist
     PERFORM Create-Queue.
* Open the queue for sending
     PERFORM Open-Queue-Send.
* Send 6 transacted, authenticated messages to the queue
     PERFORM Get-Security-Context.
     PERFORM Begin-Transaction.
     PERFORM Send-Message
          VARYING MSG-COUNTER FROM 1 BY 1 UNTIL MSG-COUNTER = 6.
     PERFORM Commit-Transaction.
     PERFORM Free-Security-Context.
* Close the queue
     PERFORM Close-Queue.
*
* Reopen the queue for receiving
     PERFORM Open-Queue-Receive.
* Receive the first message from the queue
     PERFORM Receive-Message.
* Close and delete the queue
     PERFORM Close-Queue.
     PERFORM Delete-Queue.
     MOVE ZERO TO RETURN-CODE.
     STOP RUN.
*-----------------------------------------------------------------
 Create-Queue SECTION.
*
 Create-Queue-P.
*
* Set the parameters for an MQCreateQueue call
* 1. Create a property structure including five queue properties
* 1.1 Set the queue property names in the MQ-PropID array
     MOVE PROPID-Q-PATHNAME      TO MQ-PropID(1).
     MOVE PROPID-Q-LABEL         TO MQ-PropID(2).
     MOVE PROPID-Q-TRANSACTION   TO MQ-PropID(3).
     MOVE PROPID-Q-TYPE          TO MQ-PropID(4).
     MOVE PROPID-Q-BASEPRIORITY  TO MQ-PropID(5).
*
* 1.2 Set the property values in the MQ-PropVar array
```

```
     MOVE VT-LPWSTR          TO  MQ-VARTYPE(1).
     CALL INTRINSIC ".LOC." USING Q-PATH-STRING
          GIVING MQ-LPWSTR(1).
*
     MOVE VT-LPWSTR          TO  MQ-VARTYPE(2).
     CALL INTRINSIC ".LOC." USING Q-LABEL-STRING
          GIVING MQ-LPWSTR(2).
*
     MOVE VT-UI1             TO MQ-VARTYPE(3).
     MOVE MQ-TRANSACTIONAL   TO MQ-BVAL(3).
*
     MOVE VT-CLSID           TO  MQ-VARTYPE(4).
     CALL INTRINSIC ".LOC." USING MQ-QTYPE-TEST
          GIVING MQ-PUUID(4).
*
     MOVE VT-I2              TO  MQ-VARTYPE(5).
     MOVE -2                 TO  MQ-IVAL(5).
*
* 1.3 Set the total number of active properties in the property
structure
     MOVE  5    TO cProp.
*
* 2. Set the queue path name and label
     STRING '.\HP3000-transact' LOW-VALUE
                 DELIMITED BY SIZE INTO Q-PATH-STRING.
     STRING 'HP3000 Test Queue' LOW-VALUE
                DELIMITED BY SIZE INTO Q-LABEL-STRING.
*
* 3. Assign a buffer for the queue format name (output)
     CALL INTRINSIC ".LEN." USING FormatName
                             GIVING FormatName-Length.
*
* Call the MQCreateQueue API function to create the queue
     CALL  'MQCreateQueue' USING
                                  0
                      BY REFERENCE Props
                                   FormatName
                                   FormatName-Length
                             GIVING MQ-Result.
     IF MQ-Result = MQ-OK
        GO TO Create-Queue-Exit
     ELSE
        IF MQ-Result-Flags = MQ-Error
        AND MQ-Result-Seq = MQ-ERROR-QUEUE-EXISTS
           PERFORM Path-To-FormatName
        ELSE
           DISPLAY 'MQCreateQueue ' ERR-MSG MQ-Result-Seq
```

```
           STOP RUN
         END-IF
     END-IF.
*
 Create-Queue-Exit.
         EXIT.
*-----------------------------------------------------------------
 Path-To-FormatName SECTION.
*
 Path-To-FormatName-P.
*
* If a queue with the given path name already exists, call
* MQPathNameToFormatName to retrieve its format name
     CALL  'MQPathNameToFormatName' USING
                BY REFERENCE Q-PATH-STRING
                             FormatName
                             FormatName-Length
                    GIVING MQ-Result.
     EVALUATE MQ-Result-Seq = MQ-OK
         WHEN TRUE   GO TO Path-To-FormatName-Exit
         WHEN OTHER  DISPLAY 'MQPathNameToFormatName ' ERR-MSG
                             MQ-Result-Seq
                     STOP RUN
     END-EVALUATE.
*
 Path-To-FormatName-Exit.
          EXIT.
*-----------------------------------------------------------------
 Open-Queue-Send SECTION.
*
 Open-Queue-Send-P.
*
* Call MQOpenQueue to open the queue for sending
     CALL   'MQOpenQueue' USING
                  BY REFERENCE FormatName
                               2
                               0
                  BY REFERENCE Queue-Handle
                         GIVING MQ-Result.
     EVALUATE MQ-Result-Seq = MQ-OK
         WHEN TRUE   GO TO Open-Queue-Send-Exit
         WHEN OTHER  DISPLAY 'MQOpenQueue (SEND)' ERR-MSG
                             MQ-Result-Seq
                     STOP RUN
     END-EVALUATE.
*
   Open-Queue-Send-Exit.
```

```
          EXIT.
*-----------------------------------------------------------------
 Open-Queue-Receive SECTION.
*
 Open-Queue-Receive-P.
*
* Call MQOpenQueue to open the queue for receiving
     CALL   'MQOpenQueue' USING
                  BY REFERENCE FormatName
                               1
                               1
                  BY REFERENCE Queue-Handle
                        GIVING MQ-Result.
     EVALUATE MQ-Result-Seq = MQ-OK
         WHEN TRUE   GO TO Open-Queue-Receive-Exit
         WHEN OTHER  DISPLAY 'MQOpenQueue (RECV) ' ERR-MSG
                             MQ-Result-Seq
                     STOP RUN
     END-EVALUATE.
*
   Open-Queue-Receive-Exit.
          EXIT.
*-----------------------------------------------------------------
 Get-Security-Context SECTION.
*
* Retrieve security information needed to authenticate messages
* using an internal (MSMQ) certificate. The certificate must
* be registered for the current user on the Envoy Message Queuing
Connector (MQC) machine.
*
 Get-Security-Context-P.
*
     CALL   'MQGetSecurityContext' USING
                                       0
                                       0
                           BY REFERENCE SecContext-Handle
                                 GIVING MQ-Result.
     EVALUATE MQ-Result-Seq = MQ-OK
         WHEN TRUE   GO TO Get-Security-Context-Exit
         WHEN OTHER  DISPLAY 'MQGetSecurityContext ' ERR-MSG
                             MQ-Result-Seq
                     STOP RUN
     END-EVALUATE.
*
 Get-Security-Context-Exit.
          EXIT.
*-----------------------------------------------------------------
```

```
 Free-Security-Context SECTION.
*
 Free-Security-Context-P.
*
     CALL   'MQFreeSecurityContext'
                            USING SecContext-Handle.
*
 Free-Security-Context-Exit.
          EXIT.
*-----------------------------------------------------------------
 Begin-Transaction SECTION.
*
 Begin-Transaction-P.
*
* Begin a transaction
     CALL   'MQBeginTransaction' USING
                  BY REFERENCE pTransaction
                        GIVING MQ-Result.
     EVALUATE MQ-Result-Seq = MQ-OK
         WHEN TRUE   GO TO Begin-Transaction-Exit
         WHEN OTHER  DISPLAY 'MQBeginTransaction ' ERR-MSG
                             MQ-Result-Seq
                     STOP RUN
     END-EVALUATE.
*
 Begin-Transaction-Exit.
          EXIT.
*-----------------------------------------------------------------
 Send-Message SECTION.
*
 Send-Message-P.
*
* Send a message and ask MSMQ to authenticate it.
*
* 1. Create a property structure including four message properties
* 1.1 Set the strings for the message body and label properties
*     (The message body is 'Message number <i>'. The message label
*     is 'Message label <i>'.)
*
        STRING MSG-BODY-PREFIX MSG-COUNTER
                        DELIMITED BY SIZE INTO MSG-BODY-STRING.
        STRING MSG-LABEL-PREFIX MSG-COUNTER LOW-VALUE
                        DELIMITED BY SIZE INTO MSG-LABEL-STRING.
*
* 1.2 Set the total number of active properties in the property
structure
        MOVE 4 TO cProp.
```

```
*
* 1.3 Set the aPropID array containing the message property
identifiers
        MOVE PROPID-M-BODY              TO MQ-PropID(1).
        MOVE PROPID-M-LABEL             TO MQ-PropID(2).
        MOVE PROPID-M-AUTH-LEVEL        TO MQ-PropID(3).
        MOVE PROPID-M-SECURITY-CONTEXT  TO MQ-PropID(4).
*
* 1.4 Set the aPropVar array containing the property values
        MOVE VT-VECTOR-UI1    TO MQ-VARTYPE(1).
        MOVE 50               TO MQ-CAUB-CELEMS(1).
        CALL INTRINSIC ".LOC." USING MSG-BODY-STRING
             GIVING MQ-CAUB-PELEMS(1).
*
        MOVE VT-LPWSTR        TO MQ-VARTYPE(2).
        CALL INTRINSIC ".LOC." USING MSG-LABEL-STRING
             GIVING MQ-LPWSTR(2).
*
        MOVE VT-UI4                    TO  MQ-VARTYPE(3).
        MOVE MQMSG-AUTH-LEVEL-ALWAYS   TO  MQ-ULVAL(3).
*
        MOVE VT-UI4            TO  MQ-VARTYPE(4).
        MOVE SecContext-Handle TO  MQ-ULVAL(4).
*
* Call MQSendMessage to send the message
      CALL  'MQSendMessage' USING
                                \Queue-Handle\
                  BY REFERENCE Props
                                \pTransaction\
                          GIVING MQ-Result.
      EVALUATE MQ-Result-Seq = MQ-OK
          WHEN TRUE   GO TO Send-Message-Exit
          WHEN OTHER  DISPLAY 'MQSendMessage ' ERR-MSG
                              MQ-Result-Seq
                      STOP RUN
      END-EVALUATE.
*
 Send-Message-Exit.
          EXIT.
*-----------------------------------------------------------------
 Receive-Message SECTION.
*
 Receive-Message-P.
*
* Receive a message (not as part of a transaction) and check for
* authentication.
*
```

```
* Notes on the property settings:
* 1. The BODY and LABEL message properties are left unchanged
*    from the previous send operation.
*    A successful receive will place the message body into
*    MSG-BODY-STRING and the Message Label into MSG-LABEL-STRING.
*
* 2. The AUTH-LEVEL property used in the send operation is replaced
*    with the AUTHENTICATED property to enable authentication
checking.
*
* 3. The SECURITY CONTEXT property used in the send operation is
*    replaced with the LABEL-LEN property, which specifies the size
*    of the LABEL buffer
*
* Set the total number of active properties in the property structure
       MOVE 4  TO cProp.
       MOVE PROPID-M-AUTHENTICATED  TO  MQ-PropID(3).
       MOVE VT-NULL TO MQ-VARTYPE(3).
*
* Set the buffer length for the LABEL output
       MOVE PROPID-M-LABEL-LEN       TO  MQ-PropID(4).
       MOVE VT-UI4                   TO  MQ-VARTYPE(4).
       CALL INTRINSIC ".LEN." USING Q-LABEL-STRING
                             GIVING MQ-ulVal(4).
*
* Receive the message
       CALL  'MQReceiveMessage' USING
                             \Queue-Handle\
                             \MQ-INFINITE\
                             \MQ-ACTION-RECEIVE\
             BY REFERENCE Props
                             \MQ-NULL\
                             \MQ-NULL\
                             0
                             \MQ-NO-TRANSACTION\
                     GIVING MQ-Result.
       EVALUATE MQ-Result-Seq = MQ-OK
           WHEN TRUE   GO TO Authentication-Check
           WHEN OTHER  DISPLAY 'MQReceiveMessage ' ERR-MSG
                               MQ-Result-Seq
                       STOP RUN
       END-EVALUATE.
*
* Check for authentication of the message
 Authentication-Check.
       IF MQ-BVAL(3) = MQ-AUTHENTICATE
           DISPLAY AUTH-Msg
```

```
         ELSE DISPLAY NOT-AUTH-Msg
              GO TO Receive-Call-Exit.
*
 Receive-Call-Exit.
        EXIT.
*-----------------------------------------------------------------
 Close-Queue SECTION.
*
 Close-Queue-P.
*
* Close the queue
     CALL  'MQCloseQueue' USING \Queue-Handle\
                          GIVING MQ-Result.
     EVALUATE MQ-Result-Seq = MQ-OK
         WHEN TRUE   GO TO Close-Queue-Exit
         WHEN OTHER  DISPLAY 'MQCloseQueue ' ERR-MSG
                             MQ-Result-Seq
                     STOP RUN
     END-EVALUATE.
*
 Close-Queue-Exit.
          EXIT.
*-----------------------------------------------------------------
 Delete-Queue SECTION.
*
 Delete-Queue-P.
*
* Delete the queue
     CALL 'MQDeleteQueue' USING
                    BY REFERENCE FormatName
                          GIVING MQ-Result.
*
     EVALUATE MQ-Result-Seq = MQ-OK
         WHEN TRUE   GO TO Delete-Queue-Exit
         WHEN OTHER  DISPLAY 'MQDeleteQueue ' ERR-MSG
                             MQ-Result-Seq
                     STOP RUN
     END-EVALUATE.
*
 Delete-Queue-Exit.
          EXIT.
*
*-----------------------------------------------------------------
 Commit-Transaction SECTION.
*
 Commit-Transaction-P.
*
```

```
* Commit the transaction
     CALL   'FMQCommit' USING
                 BY REFERENCE pTransaction
                              0
                              0
                              0
                       GIVING MQ-Result.
     EVALUATE MQ-Result-Seq = MQ-OK
         WHEN TRUE   GO TO Commit-Transaction-Exit
         WHEN OTHER  DISPLAY 'FMQCommit ' ERR-MSG
                             MQ-Result-Seq
                     STOP RUN
     END-EVALUATE.
*
 Commit-Transaction-Exit.
          EXIT.
```

Online samples

The Envoy MQ Client software includes several online programs and source members that you can use in your COBOL applications.

The following paragraphs describe the online samples in more detail.

Copy members

The following copy members, which are located in the Envoy MQ COBOL directory, contain code for use in your applications.

FmqConst You should include the FMQCONST copy member in every Envoy MQ Client COBOL application.

This member contains definitions of MSMQ properties, named constants, and API functions. For a complete description, see *FMQCONST copy member* on page 10.

FmqPropv The FMQPROPV copy member provides a complete COBOL definition of the MSMQ propvariant data structure. For an explanation of the propvariant structure, see *Substructures of property structures* on page 25.

The member is recommended for use in programs that create property structures dynamically. For an example of its use, see the *Sample program* on page 30.

FmqLoc The FMQLOC copy member defines the data structures used in queue queries.

The member is recommended for use in programs that create the query structures dynamically. For an example, see the FMQBLOC sample program.

Sample programs

The following sample programs, which are located in the Envoy MQ COBOL directory, contain code that illustrates various messaging operations. In particular, the samples illustrate the correct syntax for each API call. You can cut and paste code from the samples, with appropriate modifications, into your programs.

FmqbDyn FMQBDYN is a sample program illustrating the dynamic creation of property structures. The program uses the FMQPROPV copy member to define the propvariant data structure.

The program illustrates most of the common messaging operations, such as:

- Creating and deleting a queue
- Converting a queue path name to a format name
- Opening and closing a queue
- Sending and receiving messages
- Working with the MSMQ message authentication service
- Working with transactions

The complete source code of this program is printed in the *Sample program* section of this chapter, page 30.

FmqbStc FMQBSTC is a sample program illustrating basic messaging operations.

The program provides examples of:

- Connecting to and disconnecting from Envoy Message Queuing Connector (MQC)
- Creating and deleting a queue
- Converting a queue path name to a format name
- Opening and closing a queue
- Sending and receiving a message

FmqbLoc FMQBLOC is a sample program that creates a queue query dynamically. The program illustrates the use of the FMQLOC copy member, and finds a queue having a specified label.

FmqbSamp FMQBSAMP contains sample API calls for a variety of messaging operations:

- Connecting to and disconnecting from a Envoy Message Queuing Connector (MQC)
- Creating and closing a cursor
- Setting and retrieving queue properties
- Retrieving machine properties

- Converting a queue handle or GUID to a format name
- Aborting a transaction
- Registering a certificate
- Retrieving Envoy MQ version information
- Using the Envoy MQ debug log

FMQBSAMP is not a complete, compilable program. Rather, it contains fragments of code illustrating the above operations.

Envoy MQ Client for HP e3000

Index